Homework 7 - File I/O & CSV

CS 1301 - Intro to Computing - Spring 2022

Important

- Due Date: **Tuesday, March 15th, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Piazza
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to enhance your understanding of File I/O and practice reading and writing to files. The homework will consist of 5 functions for you to implement. You have been given `HW07.py` skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by Ramya Subramaniam (ramyasub@gatech.edu) & Josh Tabb (jtabb6@gatech.edu)

Part 1: File I/O

For Part 1 (File I/O functions), the `.txt` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and move the file to the same folder as your `HW07.py` file. To open the `.txt` file, you can use the built-in Notepad for Windows, TextEdit for Mac, or any other text editor of your choice.

You will be working with a text file that contains data about movies. The data will contain a movie's name, lead actor, and genre. Each movie's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `movies.txt` file as an example.

```
Movie Data

Movie #1 Name
Movie #1 Lead Actor
Movie #1 Genre

Movie #2 Name
Movie #2 Lead Actor
Movie #2 Genre

.
.
.
```

Feature Actor

Function Name: featureActor()
Parameters: filename (`str`), actorName (`str`)
Returns: movieList (`list`)
Description: You're in the mood to see one of your favorite actors performing on the big screen, but you can't remember all the movies that they starred in! Write a function which will take in the name of a `.txt` file (with the format shown above), and one actor name as parameters. (This function should return a list of all the names of films from the `.txt` file which feature the actor we specified in our parameter.) If the actor is not featured in any of the films from this file, then return the string `"Actor not found"`.

```
>>> featureActor("movies.txt", "Tom Holland")
["Spider-Man: No Way Home"]
```

```
>>> featureActor("movies.txt", "Tobey Maguire")
"Actor not found"
```

Alphabet Search

Function Name: alphabetSearch()
Parameters: filename (`str`), letter (`str`)
Returns: movieDict (`dict`)
Description: A friend recently recommended a movie for you to watch, but you forgot the name of the film, except for the first letter. You decide to find out what the movie is by searching for all movies starting with a certain letter. Write a function which takes in a `.txt` file and a letter of the alphabet; this function should return a dictionary, with names of movies beginning with the specified letter as keys of this dictionary, mapping to the lead actors of the movie. If no movie in the file begins with the specified letter, return the string `"No movie tonight"`

Note: The search through the movies file should be **case insensitive**.

```
>>> alphabetSearch("movies.txt", "d")
{"Dune": "Zendaya", "Dhoom 2": "Hrithik Roshan"}
```

```
>>> alphabetSearch("movies.txt", "c")
{"Cars 2": "Owen Wilson"}
```

Favorite Films

Function Name: favFilms()
Parameters: filename (`str`), movieList (`list`)
Returns: None (`NoneType`)
Description: Inspired by reading through the `movies.txt` file, you decide that you want to write your own `.txt` file, called `favMovies.txt`. Write a function that takes in a `.txt` file and a movieList. Using the information from the `.txt` file passed in as a parameter, your function should write to a new file called `favMovies.txt`. `favMovies.txt` should be formatted exactly the same as `movies.txt`, but it should only include information about movies that are contained in movieList.

Note: Be sure that no extra newline (`\n`) character is included at the end of your `favMovies.txt` file. The order of movies in `favMovies.txt` should match the order they appear in the **movie file**.

```
>>> favFilms("movies.txt", ["Cars 2", "Shrek"])
```

Output of favMovies.txt:

```
Movie Data

Cars 2
Owen Wilson
Adventure

Shrek
Mike Myers
Comedy
```

```
>>> favFilms("movies.txt", ["Eternals"])
```

Output of favMovies.txt:

```
Movie Data

Eternals
Angelina Jolie
Action
```

Part 2: CSV

For Part 2 (CSV functions), the `.csv` file being read from will be formatted as shown below. As stated previously, be sure to download the provided file from Canvas, and move it into the same folder as your `HW07.py` file. By default, your computer will likely use Excel on Windows, or Sheets on Mac, to open the `.csv` file, but don't be alarmed: reading values from a `.csv` file is no different than a `.txt` file; the only difference lies in the formatting of the data.

You will be working with movie data. The `.csv` file will contain a movie name, an IMDB rating (out of 100), a Rotten Tomatoes rating (out of 100), and the time of the film (in minutes), separated by commas (hence the name **Comma Separated Values** file). Each data will be separated by a new-line. Below shows how the `.csv` file will be formatted. See the provided `movies.csv` file as an example. Note that the first row contains the titles for each of the columns, and does not contain any actual data.

```
movie,imdb,rotten tomatoes,time
movie1,imdb1,rottenTomatoes1,time1
movie2,imdb2,rottenTomatoes2,time2
movie3,imdb3,rottenTomatoes3,time3
.
.
.
```

Movie Night

Function Name: movieNight()
Parameters: filename (`str`), timeLimit (`int`)
Returns: movie and average rating (`tuple`)
Description: You want to figure out what movie to watch Friday night with your roommates based on the time and average rating of each movie. Write a function that finds the movie with the highest average rating and is within your time limit (inclusive). The average rating of movie is calculated by taking an average of the IMDB and Rotten Tomatoes ratings associated with that movie. Your function should return a tuple with the name of the movie (`str`) and the average rating (`float`). **The average rating is a percentage.** Assume there will be no ties. **Round the average rating to two decimal places**.

Note: If none of the movies are within your time limit, return a string that says `"Can't do movie night on Friday"` .

```
>>> movieNight('movies.csv', 120)
("Zootopia", 89.0)
```

```
>>> movieNight('movies.csv', 150)
("Spiderman: No Way Home", 90.0)
```

Movie Recommendations

Function Name: movieRecs()
Parameters: filename (`str`), interestedList (`list`), expectedRatio (`float`)
Returns: recommendations (`dict`)
Description: Your friend wants movie recommendations from you. They give you a list of movies they are interested in and an expected ratio for the movies. Your job is to go through the list of movies they're interested in and tell them which ones are well worth their time. You can do this by looking at a movie's ratio from the movie file provided. To find the ratio for a movie, divide the average of the IMDB and Rotten Tomatoes ratings for the movie by the time of the movie. If this ratio is equal to or greater than the `expectedRatio` , place this in a dictionary, which contains the movie names (`str`) as keys and the time of the movie (`int`) (in minutes) as values. Return this dictionary.

Note: If none of the movies have a ratio equal to or greater than the `expectedRatio` return `"Too many expectations"` .

```
>>> filename = 'movies.csv'
>>> interestedList = ['Maleficent','King Richard','Divergent','La La Land',
                      'Shang-Chi and the Legend of the Ten Rings', 'Roma']
>>> expectedRatio = .6
>>> movieRecs(filename, interestedList, expectedRatio)
{'La La Land': 128, 'Roma': 135,
 'Shang-Chi and the Legend of the Ten Rings': 132, 'Maleficent': 97}
```

```
>>> filename = 'movies.csv'
>>> interestedList = ['Luca', 'The Social Network', 'Titanic',
                      'Spiderman: No Way Home', 'Enola Holmes']
>>> expectedRatio = 0.65
>>> movieRecs(filename, interestedList, expectedRatio)
{'The Social Network': 120, 'Luca': 95}
```

Grading Rubric

Function	Points
featureActor()	20
alphabetSearch()	20
favFilms()	20
movieNight()	20
movieRecs()	20
Total	**100**

Provided

The `HW07.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW07.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the “Resubmit” button at the lower right-hand corner of Gradescope. You do not need to submit your `HW07.py` on Canvas.